**Practica 4:**

**Construcción de árboles de sintaxis abstracta**

**Grupo 11:**

**Youssef El Faqir El Rhazoui**

**Enrique Ávila Rodríguez**

1. **Conjunto de funciones constructoras**

**Prog:** Sec\_Dec **x** Sec\_Ins 🡪 Prog

**Sec\_Dec:** LDs 🡪 Prog

**Sec\_Ins:** LIs 🡪 Prog

**LD\_simp:** String **x** String 🡪 D

**LD\_comp:** String **x** String **x** LDs 🡪 LDs

**LI\_simp:** String **x** Exp 🡪 I

**LI\_comp:** String **x** Exp **x** LIs 🡪 LIs

**Mas:** Exp **x** Exp 🡪 Exp

**Menos:** Exp **x** Exp 🡪 Exp

**And:** Exp **x** Exp 🡪 Exp

**Or:** Exp **x** Exp 🡪 Exp

**Distinto:** Exp **x** Exp 🡪 Exp

**Igual:** Exp **x** Exp 🡪 Exp

**Menor\_que:** Exp **x** Exp 🡪 Exp

**Menor\_igual\_que:** Exp **x** Exp 🡪 Exp

**Mayor\_que:** Exp **x** Exp 🡪 Exp

**Mayor\_igual\_que:** Exp **x** Exp 🡪 Exp

**Mul:** Exp **x** Exp 🡪 Exp

**Div:** Exp **x** Exp 🡪 Exp

**Not:** Exp 🡪 Exp

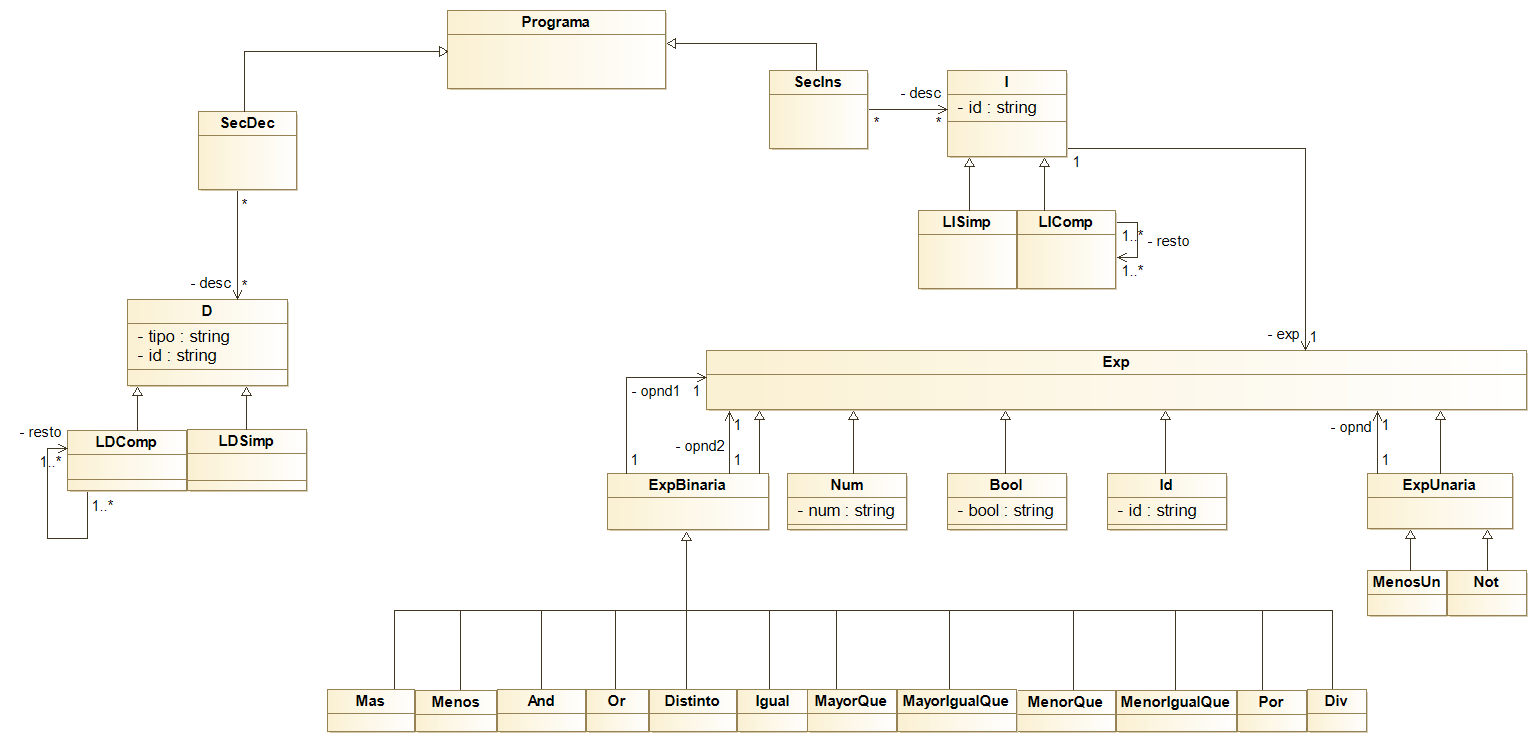
**Menos\_unario:** Exp 🡪 Exp

**Num:** String 🡪 Exp

**Bool:** String 🡪 Exp

**Id:** String 🡪 Exp

1. **Diagrama de clases**



1. **Gramática de atributos**

Prog 🡪 Sec\_Dec && Sec\_Ins

Prog.a = prog(Sec\_Dec.a, Sec\_Ins.a)

Sec\_Dec 🡪 Sec\_Dec; D

.a = ldCompuesta(D.tipo, D.id, .a)

Sec\_Dec 🡪 D

Sec\_Dec.a = ldSimple(D.tipo, D.id)

Sec\_Ins 🡪 Sec\_Ins; I

.a = liCompuesta(I.id, I.exp, .a)

Sec\_Ins 🡪 I

Sec\_Ins.a = liSimple(I.id, I.exp)

D 🡪 tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I 🡪 identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.v

Exp0 🡪 Exp0 Op0 Exp1

.v = mkexp(Op0.op, .v, Exp1.v)

Exp0 🡪 Exp1

Exp0.v = Exp1.v

Exp1 🡪 Exp2 and Exp1

.v = and(Exp2.v, .v)

Exp1 🡪 Exp2 or Exp2

Exp1.v = or(.v, .v)

Exp1 🡪 Exp2

Exp1.v = Exp2.v

Exp2 🡪 Exp3 Op2 Exp3

Exp2.v = mkexp(Op2.op, .v, .v)

Exp2 🡪 Exp3

Exp2.v = Exp3.v

Exp3 🡪 Exp3 Op3 Exp4

.v = mkexp(Op3.op, .v, Exp4.v)

Exp3 🡪 Exp4

Exp3.v = Exp4.v

Exp4 🡪 - Exp4

.v = menos\_unario(.v)

Exp4 🡪 not Exp5

Exp4.v = not(Exp5.v)

Exp4 🡪 Exp5

Exp4.v = Exp5.v

Exp5 🡪 numero

Exp5.v = num(numero.lex)

Exp5 🡪 booleano

Exp5.v = bool(booleano.lex)

Exp5 🡪 identificador

Exp5.v = id(identificador.lex)

Exp5 🡪 (Exp0)

Exp5.v = Exp0.v

Op0 🡪 +

Op0.op = “+”

Op0 🡪 -

Op0.op = “-”

Op2 🡪 !=

Op2.op = “!=”

Op2 🡪 ==

Op2.op = “==”

Op2 🡪 <

Op2.op = “<”

Op2 🡪 <=

Op2.op = “<=”

Op2 🡪 >

Op2.op = “>”

Op2 🡪 >=

Op2.op = “>=”

Op3 🡪 \*

Op3.op = “\*”

Op3 🡪 /

Op3.op = “/”

**Definimos la función mkexp como sigue:**

fun mkexp(op, opnd1,opnd2) {

switch(op) {

"+" => return suma(opnd1,opnd2)

"-" => return resta(opnd1,opnd2)

"!=" => return distinto(opnd1,opnd2)

"==" => return igual(opnd1,opnd2)

"<" => return menorQue(opnd1,opnd2)

"<=" => return menorIgualQue(opnd1,opnd2)

">" => return mayorQue(opnd1,opnd2)

">=" => return mayorIgualQue(opnd1,opnd2)

"\*" => return mul(opnd1,opnd2)

"/" => return div(opnd1,opnd2)

}

}

1. **Acondicionamiento para imp descendente**

Prog 🡪 Sec\_Dec && Sec\_Ins

Prog.a = prog(Sec\_Dec.a, Sec\_Ins.a)

Sec\_Dec 🡪 D PDec

PDec.ah = ldSimple(D.tipo, D.id)

Sec\_Dec.a = PDec.a

PDec 🡪 ; D PDec

.a = ldCompuesta(.ah, D.a)

.a = .a

PDec 🡪 ε

PDec.a = PDec.ah

Sec\_Ins 🡪 I PIns

PIns.ah = liSimple(I.id, I.exp)

Sec\_Ins.a = PIns.a

PIns 🡪 ; I PIns

.a = liCompuesta(.ah, I.a)

.a = .a

PIns 🡪 ε

PIns.a = PIns.ah

D 🡪 tipo identificador

D.tipo = tipo.lex

D.id = identificador.lex

I 🡪 identificador = Exp0

I.id = identificador.lex

I.exp = Exp0.v

Exp0 🡪 Exp1 RExp0

RExp0.vh = Exp1.v

Exp0.v = RExp0.v

RExp0 🡪 Op0 Exp1 RExp0

.vh = mkexp(Op0.op, .vh, Exp1.v)

.v = .v

RExp0 🡪 ε

RExp0.v = RExp0.vh

Exp1 🡪 Exp2 RExp1

RExp1.vh = Exp2.v

Exp1.v = RExp1.v

RExp1 🡪 and Exp1

RExp1.v = and(RExp.vh, Exp1.v)

RExp1 🡪 or Exp2

RExp1.v = or(RExp.vh, Exp2.v)

RExp1 🡪 ε

RExp1.v = RExp1.vh

Exp2 🡪 Exp3 RExp2

RExp2.vh = Exp3.v

Exp2.v = RExp2.v

RExp2 🡪 Op2 Exp3

RExp2.v = mkexp(Op2.op, RExp2.vh, Exp3.v)

RExp2 🡪 ε

RExp2.v = RExp2.vh

Exp3 🡪 Exp4 RExp3

RExp3.vh = Exp4.v

Exp3.v = RExp3.v

RExp3 🡪 Op3 Exp4 RExp3

.vh = mkexp(Op3.op, .vh, Exp4.v)

.v = .v

RExp3 🡪 ε

RExp3.v = RExp3.vh

Exp4 🡪 - Exp4

.v = menos\_unario(.v)

Exp4 🡪 not Exp5

Exp4.v = not(Exp5.v)

Exp4 🡪 Exp5

Exp4.v = Exp5.v

Exp5 🡪 numero

Exp5.v = num(numero.lex)

Exp5 🡪 booleano

Exp5.v = bool(booleano.lex)

Exp5 🡪 identificador

Exp5.v = id(identificador.lex)

Exp5 🡪 (Exp0)

Exp5.v = Exp0.v

Op0 🡪 +

Op0.op = “+”

Op0 🡪 -

Op0.op = “-”

Op2 🡪 !=

Op2.op = “!=”

Op2 🡪 ==

Op2.op = “==”

Op2 🡪 <

Op2.op = “<”

Op2 🡪 <=

Op2.op = “<=”

Op2 🡪 >

Op2.op = “>”

Op2 🡪 >=

Op2.op = “>=”

Op3 🡪 \*

Op3.op = “\*”

Op3 🡪 /

Op3.op = “/”